MEMORIA PARTE 1

GRUPO 9: Artem Vartanov y Daniel Coleto Quereda

**Ejercicio 1**

b.1) Inicializa el problema con el siguiente estado inicial: (2,5,6,0,3,8,4,1,7). Para comprobar que lo has hecho bien.

¿Qué sentencia tienes que poner para saber el estado inicial?

state = (2,5,6,0,3,8,4,1,7)  
ep = EightPuzzle(state)  
print(ep.initial) # devuelve la tupla del estado inicial

¿Cuál es el resultado de ejecutar esta sentencia?

(2, 5, 6, 0, 3, 8, 4, 1, 7)

b.2)

¿Qué sentencias necesitamos para ejecutar las acciones desde el estado inicial? Indica las sentencias realizadas y el resultado de estas acciones.

# devuelve acciones posibles a realizar(adonde se puede mover el "agujero")  
posibles = ep.actions(ep.initial)

print(posibles)

Resultado:

['UP', 'DOWN', 'RIGHT']

b.3) ¿Qué sentencia necesitamos para comprobar el objetivo? Indica la sentencia para comprobar el objetivo del estado inicial y el resultado de su ejecución.

print(ep.goal\_test(ep.initial))

Resultado:

False

b.4) Ejecuta la acción “UP” y comprueba si el estado obtenido es el objetivo.

# Para ejecutar las acciones necesitamos ejecutar la funcion result

state = ep.result(state, "UP")

print(state)

print(ep.goal\_test(state))

Resultado:

(0, 5, 6, 2, 3, 8, 4, 1, 7)

False

**Ejercicio 2**

1. Modeliza el problema de la forma que hemos visto en clase. Recuerda que tienes que definir cómo son los estados, cuál es el estado inicial, cuáles son las acciones , cómo comprobar el objetivo y cuál es el coste de cada acción. Se recomienda que el estado inicial sea el tablero sin reinas.

**EXPLICACION DE ALGORITMO (PARA 4 REINAS)**

El estado inicial(state) es el tablero sin reinas:

[0, [-1, -1, -1, -1], [False, False, False, False, False, False, False], [False, False, False, False, False, False, False]]

0 – La columna en la que vamos a colocar la siguiente reina

[-1, -1, -1, -1] – Un elemento para cada fila. Al acceder por indice i, nos devuelve la columna en la que está la reina que está en la fila i

El primer vector de booleans – Para las diagonales “/” (pendiente creciente)

El segundo vector de booleans – Para las diagonales “\” (pendiente decreciente)

Para cada vector de booleans, el elemento en el indice i nos dice que esa diagonal ya contiene una reina

Al colocar una reina, en la posición fila = i, columna = state[0], asignamos state[1][i] = columna y incrementamos una unidad state[0]

A continuación viene una explicación del almacenamiento de las diagonales

Sea un tablero 4x4:

-----------------------------

| 0,0 | 0,1 | 0,2 | 0,3 |

-----------------------------

| 1,0 | 1,1 | 1,2 | 1,3 |

-----------------------------

| 2,0 | 2,1 | 2,2 | 2,3 |

-----------------------------

| 3,0 | 3,1 | 3,2 | 3,3 |

-----------------------------

i -> indice del primer vector de booleans que guarda si hay una reina en esa diagonal o no

i 0 1 2 3 4 5 6

-----------------------------

| 0,0 | 0,1 | x | 0,3 |

-----------------------------

| 1,0 | x | 1,2 | 1,3 |

-----------------------------

| x | 2,1 | 2,2 | 2,3 |

-----------------------------

| 3,0 | 3,1 | 3,2 | 3,3 |

---------------------------

La diagonal marcada por x se guardaria en state[2][2] (i = 2)

Porque la suma de posX + posY para cada elemento de diagonal es igual a 2:

2 + 0 = 2;

1 + 1 = 2;

0 + 2 = 2;

Por tanto se necesitarían tam \* 2 - 1 => 4 \* 2 - 1 = 7 posiciones para tener una para cada diagonal. Pero nos faltan las otras diagonales.

j -> indice del segundo vector de booleans que guarda si hay una reina en esa diagonal o no

6

5

4

-------------------------------

3 | x | 0,1 | 0,2 | 0,3 |

-------------------------------

2 | 1,0 | x | 1,2 | 1,3 |

-------------------------------

1 | 2,0 | 2,1 | x | 2,3 |

-------------------------------

0 | 3,0 | 3,1 | 3,2 | x |

------------------------------

La diagonal marcada por x se guardaría en state[3][3] (j = 3)

Porque la suma de tam - 1 - posX + posY para cada elemento de diagonal es igual a 3:

4 - 1 - 0 + 0 = 3;

4 - 1 - 1 + 1 = 3;

4 - 1 - 2 + 2 = 3;

4 - 1 - 3 + 3 = 3;

Espacio de estados más abajo.

Espacio de estados:

[0, [-1, -1, -1, -1], [False, False, False, False, False, False, False], [False, False, False, False, False, False, False]]

[1, [0, -1, -1, -1], [True, False, False, False, False, False, False], [False, False, False, True, False, False, False]]

[2, [0, -1, 1, -1], [True, False, False, True, False, False, False], [False, False, True, True, False, False, False]]

[2, [0, -1, -1, 1], [True, False, False, False, True, False, False], [False, True, False, True, False, False, False]]

[3, [0, 2, -1, 1], [True, False, False, True, True, False, False], [False, True, False, True, True, False, False]]

[1, [-1, 0, -1, -1], [False, True, False, False, False, False, False], [False, False, True, False, False, False, False]]

[2, [-1, 0, -1, 1], [False, True, False, False, True, False, False], [False, True, True, False, False, False, False]]

[3, [2, 0, -1, 1], [False, True, True, False, True, False, False], [False, True, True, False, False, True, False]]

[4, [2, 0, 3, 1], [False, True, True, False, True, True, False], [False, True, True, False, True, True, False]]

[1, [-1, -1, 0, -1], [False, False, True, False, False, False, False], [False, True, False, False, False, False, False]]

[2, [1, -1, 0, -1], [False, True, True, False, False, False, False], [False, True, False, False, True, False, False]]

[3, [1, -1, 0, 2], [False, True, True, False, False, True, False], [False, True, True, False, True, False, False]]

[4, [1, 3, 0, 2], [False, True, True, False, True, True, False], [False, True, True, False, True, True, False]]

[1, [-1, -1, -1, 0], [False, False, False, True, False, False, False], [True, False, False, False, False, False, False]]

[2, [1, -1, -1, 0], [False, True, False, True, False, False, False], [True, False, False, False, True, False, False]]

[3, [1, -1, 2, 0], [False, True, False, True, True, False, False], [True, False, False, True, True, False, False]]

[2, [-1, 1, -1, 0], [False, False, True, True, False, False, False], [True, False, False, True, False, False, False]]

Las acciones son las filas en las que podemos poner la reina actual. En el caso de que es una lista vacía, no podemos realizar ninguna acción. La pre-condición es que el estado es válido, reinas no se comen entre si. La post-condición es que el estado es válido, reinas no se comen entre si.

La comprobación del objectivo compara si state[0] == tamaño del tablero. Si es así hemos rellenado todas las columnas y alcanzado el objetivo. La pre-condición es que el estado es válido, reinas no se comen entre si. La post-condición es que el estado es válido, reinas no se comen entre si.

Nota: Los costes de cada función está puestas en el código.

